UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/779,328	02/13/2004	Umesh Madan	MS1-1851US	5250

22801 7590 11/21/2007

LEE & HAYES PLLC
421 W RIVERSIDE AVENUE SUITE 500
SPOKANE, WA 99201

EXAMINER	
TSAI, SHENG JEN	
ART UNIT	**PAPER NUMBER**
2186	

MAIL DATE	DELIVERY MODE
11/21/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)
	10/779,328	MADAN ET AL.
	Examiner	Art Unit
	Sheng-Jen Tsai	2186

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 12 September 2007.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-38 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-38 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on 13 February 2004 is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office Action is taken in response to Applicants' Amendments and Remarks filed on September 12, 2007 regarding application 10,779,328 filed on February 13, 2004.

2. Claims 1, 8 and 17-27 have been amended.

Claims 1-38 are pending for consideration.

3. ***Response to Remarks and Amendments***

Applicants' amendments and remarks have been fully and carefully considered with examiner's response set forth below.

In view of the amendments, rejections of claims 1-7 under 35 U.S.C. 112, second paragraph, have been withdrawn.

Independent claims 1 and 17 have been amended to include, in addition to those previously presented, the new limitation that the inverse query engine cache also comprises a control module, a cache, an expiration module, a trim module, a cache weight module, a cache weight, an optimal weight, and a maximum weight. It is note that these additional elements are also taught in combination by the previously relied on references (Schneider, US 5,668,987 and Chen et al., US 2004/0001498). Another iteration of claim analysis based on the previously relied on references has been made. Refer to the corresponding sections of the following claim analysis for details.

Applicants contend that it is improper to combine the references of Schneider and Chen because of the lack of motivation to combine them. The Examiner would like to remind Applicants, by referring to the recent ruling of KSR Int'l Co. v. Teleflex Inc.,

127 S. Ct. 1727, 1734, 82 USPQ2d 1385, 1391 (2007) by the Supreme Court, that the most important question to be considered in this case is whether the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains:

"Section 103 forbids issuance of a patent when 'the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains.'" KSR Int'l Co. v. Teleflex Inc., 127 S. Ct. 1727, 1734, 82 USPQ2d 1385, 1391 (2007). The question of obviousness is resolved on the basis of underlying factual determinations including (1) the scope and content of the prior art, (2) any differences between the claimed subject matter and the prior art, and (3) the level of skill in the art. Graham v. John Deere Co., 383 U.S. 1, 17-18, 148 USPQ 459, 467 (1966). See also KSR, 127 S. Ct. at 1734, 82 USPQ2d at 1391 ("While the sequence of these questions might be reordered in any particular case, the [Graham] factors continue to define the inquiry that controls."). The Court in Graham further noted that evidence of secondary considerations, such as commercial success, long felt but unsolved needs, failure of others, etc., "might be utilized to give light to the circumstances surrounding the origin of the subject matter sought to be patented." 383 U.S. at 18, 148 USPQ at 467. "If a court, or patent examiner, conducts this analysis and concludes the claimed subject matter was obvious, the claim is invalid under § 103." KSR, 127 S. Ct. at 1734, 82 USPQ2d at 1391.

In KSR, the Supreme Court emphasized "the need for caution in granting a patent based on the combination of elements found in the prior art," id. at 1739, 82 USPQ2d at 1395, and discussed circumstances in which a patent might be determined to be obvious. In particular, the Supreme Court emphasized that "the principles laid down in Graham reaffirmed the 'functional approach' of Hotchkiss, 11 How. 248 [(1850)]." KSR, 127 S. Ct. at 1739, 82 USPQ2d at 1395 (citing Graham v. John Deere Co., 383 U.S. 1, 12, 148 USPQ 459, 464 (1966)), and reaffirmed principles based on its precedent that "[t]he combination of familiar elements according to known methods is likely to be obvious when it does no more than yield predictable results." Id. The operative question in this "functional approach" is thus "whether the improvement is more than the predictable use of prior art elements according to their established functions." Id. at 1740, 82 USPQ2d at 1396.

4. ***Claim Rejections - 35 USC § 102***

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

6. Claims 32-33 and 37 are rejected under 35 U.S.C. 102(b) as being anticipated by Schneider (US 5,668,987).

As to claim 32, Schneider discloses **an inverse query engine having an integrated cache** [figure 2, 260 shows the inverse query engine; figure 3B shows a subquery cache (340) which is part of the query engine].

As to claim 33, Schneider teaches that **the inverse query engine as recited in claim 32, wherein the cache is bound to a finite size** [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 37, Schneider teaches **the inverse query engine as recited in claim 32, wherein the cache is maintained within predefined limits by removing at least one filter from a group of filters stored in the cache that has been used less recently than other filters in the group of filters** [Finally, an "LRU Depth" of the deepest hit in the cache is also tracked. Since the cache can adjust its size dynamically, it itself has a "depth"--that is, how many rows comprise the cache at a given instance. An "LRU Depth" variable is therefore employed to track how deep (i.e., the deepest row) into the cache there has been a "hit" (column 2, lines 20-40); generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)].

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

8. Claims 1-31, 34-36 and 38 are rejected under 35 U.S.C. 103(a) as being unpatentable over Schneider (US 5,668,987), and in view of Chen et al. (US 2004/0001498, hereinafter referred to as Chen).

As to claim 1, Schneider and Chen disclose **a method, comprising**:

receiving a request to add a new filter [Schneider: After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its input value(s)/result value pair is added to the top (i.e., first row) of the subquery cache (column 7, lines 28-33); Chen also teaches receiving a request to add a new filter (a router sends notification requesting filters, paragraph 0011)] **to a filter table** [Schneider: figure 3A shows examples of filter tables TABLE T1 (310) and TABLE T2 (320); Chen shows it in figure 18] **stored in an inverse query engine cache** [Schneider: figure 2, 260 shows the inverse query engine; figure 3B shows a subquery cache (340) which is part of the query engine; Chen shows it in figures 7 and 8];

adding the new filter to the filter table [Schneider: After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its input value(s)/result value pair is added to the top (i.e., first row) of the subquery cache (column 7, lines 28-33); Chen: abstract: a plurality of filters are received], **wherein the new filter comprises a condition field** [Chen: paragraph 0156: 5) filters may involve simple (e.g., >, ==, <=, etc.) or complicated tests or operators (e.g., membership test, regular expression, string matching, range operators such as intersection queries, enclosure queries, containment queries etc.)], **a data field** [Schneider: because each filter contained in the filter tables (Tables T1 and T2, figure 3A, 310 and 320) has a data

field; [Chen: paragraph 0156: 4) filters may involve simple (integer, Boolean) or complicated data type], **an expiration time field** [Chen: paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired], **a filter weight field** [Chen: paragraph 0156: 3) filters may involve fixed size or variable-size attributes], **and a permanent flag field** [Chen: paragraph 0178: There are cases in some publish-subscribe network strategy (e.g., CBT) where filters have no expiration time-outs, and once they have been propagated to one node, they will remain persistent over that link until some changes to the filter set (e.g., filter addition, filter removal, etc.) take place];

determining the filter table of a bounded size [Schneider: While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)];

maintaining the inverse query engine cache at or below a maximum cache size, wherein the size of the inverse query engine cache may be indicated by size of the filter table, estimate of size of the filter table, or by cache usage [Schneider: While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)) by trimming old, least used items from the

table (generally cache rows are filled on a least-recently used_(LRU) basis (column 7, lines 26-28)], **wherein the inverse query engine cache comprises a control module** [Schneider: the engine, figure 2, 260; Chen: the intelligent router, figure 4, 92], **a cache** [it is inherent that the query engine cache has a cache, otherwise it will not be a cache]**, an add filter module** [Schneider: While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are <u>added to the cache</u> (column 8, lines 26-29)]**, a remove filter module** [Schneider teaches any unique items (i.e., values) are added to the cache (column 8, lines 26-29)) by <u>trimming old, least used items from the table</u> (generally cache rows are filled on a <u>least-recently used (LRU) basis</u> (column 7, lines 26-28); Chen: paragraph 0178: filter removal]**, a matcher** [Schneider: The principle advantage is if a <u>cache hit</u> (i.e., <u>lookup with appropriate match</u>) occurs, the system has saved time which would otherwise be required for executing the subquery to resolve a subquery result value (column 6, lines 49-53); Chen: All notifications <u>matching the subscription</u> can be delivered to the subscriber via a callback or other type of function that the subscriber provides at the time it registers its subscription or at other times. One subscription can be broken down into many filters (paragraph 0039)], **a maintainer** [Schneider: While the number of items is less than the cache size (step 401), the method continues to execute and <u>maintain the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]**, an expire module** [Chen: paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration

of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired], **a trim module** [Schneider teaches any unique items (i.e., values) are added to the cache (column 8, lines 26-29)) by trimming old, least used items from the table (generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28); Chen: paragraph 0178: filter removal], **a cache weight module** [Schneider: In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29); Chen: paragraph 0156: 3) filters may involve fixed size or variable-size attributes], **a cache weight** [Schneider: In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29); Chen: paragraph 0156: 3) filters may involve fixed size or variable-size attributes], **an optimal weight** [Schneider: In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the

method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)], **a maximum weight** [Schneider: In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)], **a filter table** [Schneider: the query engine (figure 2, 260) contains filter tables (Tables, figure 2, 250; Tables T1 and T2, figure 3A, 310 and 320); Chen: abstract: a filter receiving module, a filter reduction module, and a filter propagation module], **a most recently used list** [Chen: Intelligent router 92 time marks the data (step 324) and locally caches it such as in memory 94 or secondary storage 97 (step 326). It indexes the cached data by, for example, channel ID, subjects, and time stamps (step 328) (paragraph 0100); It should be noted that the time stamps information provides directly the most recently used as well as the least recently used attributes; Schneider: generally cache rows are filled on a least-recently used_(LRU) basis (column 7, lines 26-28)], **and an expiration list** [Chen: paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired];

wherein the expiration list comprises a filter identifier including an expiration value in an expiration field [Chen: paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired];

removing a filter based on an expiration time [Chen: paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired; thus a filter is removed (i.e. not propagated any further) if it has expired];

trimming the filter table [Schneider teaches maintaining the inverse query engine cache at or below a maximum cache size as new items are added to the filter table (While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)) by trimming old, least used items from the table (generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28); Chen: paragraph 0178: filter removal]; **and**

wherein the inverse query engine cache is used exclusively by an inverse query engine to store filters associated therewith [Schneider: Database system and methods are described for improving execution speed of database queries (e.g., for decision support) by optimizing execution of nested queries or "subqueries," such as are commonly used in client/server database environments. In particular, a subquery

cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract)].

Regarding claim 1, Schneider teaches that the new filter comprises a data field, but does not teach a condition field, an expiration time field, a filter weight field, or a permanent field.

However, Chen's filters comprise all these fields: **a condition field** [Chen: paragraph 0156: 5) filters may involve simple (e.g., >, ==, <=, etc.) or complicated tests or operators (e.g., membership test, regular expression, string matching, range operators such as intersection queries, enclosure queries, containment queries etc.)]**, a data field** [Schneider: because each filter contained in the filter tables (Tables T1 and T2, figure 3A, 310 and 320) has a data field; Chen: paragraph 0156: 4) filters may involve simple (integer, Boolean) or complicated data type]**, an expiration time field** [Chen: paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired]**, a filter weight field** [Chen: paragraph 0156: 3) filters may involve fixed size or variable-size attributes]**, and a permanent flag field** [Chen: paragraph 0178: There are cases in some publish-subscribe network strategy (e.g., CBT) where filters have no expiration time-outs, and once they have been propagated to one node, they will remain persistent over that link until some changes to the filter set (e.g., filter addition, filter removal, etc.) take place].

Therefore, it would have been obvious for one of ordinary skills in the art to include these fields as filter attributes, as demonstrated by Chen, and to incorporate it into the existing system disclosed by Schneider because Chen teaches that doing so would allow facilitating propagating filters efficiently, alleviate the burden on processing elements, and add values to the applications [paragraph 0008].

Regarding claim 1, Schneider does not explicitly mention **an expire module**, **an expiration list**, and **removing a filter based on an expiration item.**

However, Chen teaches these limitations [paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired; paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired; thus a filter is removed (i.e. not propagated any further) if it has expired].

Therefore, it would have been obvious for one of ordinary skills in the art to recognize the need to remove the contents of a memory block when the size of the memory is limited, as demonstrated by both Schneider and Chen, and that removing the contents of a memory block based on an expiration time is a straightforward and efficient way of achieving this goal, as demonstrated by Chen, and to incorporate it into the existing system disclosed by Schneider, to further enhances the cache

replacement strategy beyond the commonly adopted "Least Recently Used" scheme, and makes better utilization of the precious resource of cache capacity.

Regarding claim 1, Schneider does not explicitly mention the use of **a most recently used list**.

However, Schneider does teaches the use of Least Recent Used method for removing an item from the filter table [generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)].

Further, Chen teaches time marks the data to be cached [Intelligent router 92 time marks the data (step 324) and locally caches it such as in memory 94 or secondary storage 97 (step 326). It indexes the cached data by, for example, channel ID, subjects, and time stamps (step 328) (paragraph 0100)]. It should be noted that the time stamps information provides directly the most recently used as well as the least recently used attributes.

Therefore, it would have been obvious for one of ordinary skills in the art to include the time stamps information, as demonstrated by Chen, and to incorporate it into the existing system disclosed by Schneider, so that the cache controller would have the most recently used information in addition to the least recently used information, which would further enhance the replacement/removing scheme of a filter entry.

As to claim 2, Schneider teaches that **the method as recited in claim 1, further comprising maintaining the size of the inverse query engine cache between an optimal cache size and the maximum cache size** [In particular, a subquery cache is

provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 3, Schneider teaches that **the method as recited in claim 1, wherein the maintaining further comprises**:

determining if the addition of the new filter to the new filter table increases the cache size above the maximum cache size [figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row]; **and**

removing one or more filters from the filter table if the addition of the new filter causes the cache size to exceed the maximum cache size [For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)].

As to claim 4, Schneider teaches that **the method as recited in claim 3, wherein the determining step further comprises**:

determining a relative size of the new filter [figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row];

assigning a filter weight to the new filter based on the relative filter size [the corresponding weight is the size of the filter item; figure 4A, step 401, "while items <

cache rows" determines if the size of the items to be added would exceed the size of a cache row];

deriving a cache weight by summing filter weights of all filters in the filter table including the new filter [the corresponding cache weight is the sum of the size of all filter items currently stored in the cache; While the number of items is less than the cache size (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]; **and**

comparing the cache weight to the maximum cache size [After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its <u>input value(s)/result value pair</u> is <u>added</u> to the top (i.e., first row) of the subquery cache (column 7, lines 28-33); While the number of items is less than the cache size (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 5, Schneider teaches that **the method as recited in claim 1, wherein the maintaining further comprises**:

identifying a weight associated with the new filter [the corresponding weight is the size of the filter item; figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row];

adding the weight associated with the new filter to a cache weight that is the sum of filter weights of filters in the filter table, each filter having a filter weight [the corresponding cache weight is the sum of the size of all filter items currently stored

in the cache; After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its input value(s)/result value pair is added to the top (i.e., first row) of the subquery cache (column 7, lines 28-33); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]; **and**

comparing the cache weight to the maximum cache size [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 6, Chen teaches that **the method as recited in claim 1, further comprising identifying one or more expired filters in the filter table; and wherein the maintaining the inverse query engine cache further comprises removing at least one of the identified expired filters** [paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired; thus a filter is removed (i.e. not propagated any further) if it has expired; paragraph 0178: filter removal].

As to claim 7, Schneider teaches that **the method as recited in claim 1, further comprising a least recently used filter in the filter table** [For the latter case (i.e., a

cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)]; **and**

wherein the maintaining the inverse query engine cache further comprises removing the least recently used filter from the filter table when a size of the inverse query engine cache reaches the maximum cache size [Finally, an "LRU Depth" of the deepest hit in the cache is also tracked. Since the cache can adjust its size dynamically, it itself has a "depth"--that is, how many rows comprise the cache at a given instance. An "LRU Depth" variable is therefore employed to track how deep (i.e., the deepest row) into the cache there has been a "hit" (column 2, lines 20-40)].

As to claim 8, refer to "As to claim 1" presented earlier in this Office Action.

As to claim 9, Chen teaches that **the system as recited in claim 8, further comprising an expiration module configured to remove expired filters from the filter table** [paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired; thus a filter is removed (i.e. not propagated any further) if it has expired; paragraph 0178: filter removal].

As to claim 10, Chen teaches that **the system as recited in claim 9, wherein the maintainer is further configured to invoke the expiration module when a new filter is added to the filter table** [paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph

0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired; thus a filter is removed (i.e. not propagated any further) if it has expired; paragraph 0178: filter removal].

As to claim 11, Schneider teaches that **the system as recited in claim 8, further comprising a trim module configured to remove one or more filters from the filter table when the cache reaches a maximum cache size** [While the number of items is less than the cache size (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29); Finally, an "LRU Depth" of the deepest hit in the cache is also tracked. Since the cache can adjust its size dynamically, it itself has a "depth"--that is, how many rows comprise the cache at a given instance. An "LRU Depth" variable is therefore employed to track how deep (i.e., the deepest row) into the cache there has been a "hit" (column 2, lines 20-40)].

As to claim 12, Schneider teaches that **the system as recited in claim 11, wherein the trim module is further configured to remove the one or more filters from the filter table until the cache is reduced to an optimal cache size** [In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving <u>an optimal cache size</u> (abstract); For the latter case (i.e., a cache employing multiple rows or entries),

generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)].

As to claim 13, Chen teaches **that the system as recited in claim 11, wherein the trim module is further configured to determine if a permanent flag in a filter is set and, if the permanent flag is set, to leave the filter in the filter table** [paragraph 0178: There are cases in some publish-subscribe network strategy (e.g., CBT) where filters have no expiration time-outs, and once they have been propagated to one node, they will remain persistent over that link until some changes to the filter set (e.g., filter addition, filter removal, etc.) take place].

As to claim 14, Schneider teaches that **the system as recited in claim 11, wherein**:

a filter weight is associated with each filter in the filter table [the corresponding weight is the size of the filter item; figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row];

the cache size further comprises a cache weight that is a sum of all filter weights in the filter table [the corresponding cache weight is the sum of the size of all filter items currently stored in the cache; While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)];

the maximum cache size further comprises a maximum cache weight [the corresponding cache weight is the sum of the size of all filter items currently stored in the cache; While the number of items is less than the cache size (step 401), the

method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]; **and wherein the trim module is further configured to deduct a filter weight from the cache weight when a filter associated with the filter weight is removed from the filter table** [For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28); since the weight is the size, whenever a filter is removed from the cache, the size of the cache decreases by the size of the filter, so as the weight].

As to claim 15, Schneider teaches that **the system as recited in claim 14, further comprising a cache weight module configured to assign a filter weight to each filter added to the filter table, each filter weight identifying a relative size of a filter with regard to other filters in the filter table** [since the weight is the size, whenever a filter is added to the cache, the size of the cache increases by the size of the filter, so as the weight].

As to claim 16, Schneider in view of Chen teaches **the system as recited in claim 11, further comprising: a most recently used list configured to track usage of filter table filters according to when filters are used, and indicating that a filter has been most recently used when the filter is the latest filter to match an input** [Chen: Intelligent router 92 time marks the data (step 324) and locally caches it such as in memory 94 or secondary storage 97 (step 326). It indexes the cached data by, for example, channel ID, subjects, and time stamps (step 328) (paragraph 0100); It should

be noted that the time stamps information provides directly the most recently used as well as the least recently used attributes]; **and**

wherein the trim module is further configured to remove the one or more filters from the filter table based on which filters have been least recently used [Schneider: For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)].

As to claim 17, Schneider discloses **one or more computer-readable storage media** [main memory, figure 1A, 102; mass storage, figure 1A, 107] **storing computer-executable instructions** [program code, columns 13~66] **that, when executed on a computer** [central processor, figure 1A, 101], **perform the following steps**:

receiving a request to add a new query to an inverse query engine cache that stores multiple queries, each query having a query size associated therewith [After the cache is scanned and a "miss" occurs (i.e., no match), the subquery is executed and its input value(s)/result value pair is added to the top (i.e., first row) of the subquery cache (column 7, lines 28-33); figure 4A, step 401, “while items < cache rows” determines if the size of the items to be added would exceed the size of a cache row];

defining conditions and processing input that satisfies the conditions [figure 3A shows an example of the conditions: Select C1 from T1 where C2=(Select MAX(C3) from T2 where T2.C4=T1.C1];

deriving a cache size that is a sum of query sizes of the queries stored in the inverse query engine [In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract)];

determining if the cache size is at greater than or equal to a maximum cache size, where the cache size may be determined comprising cache usage, size of the query, or estimate of size of query [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29); In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract)];

removing one or more queries from the inverse query engine cache if the cache size is greater than or equal to the maximum cache size [For the latter case (i.e., a cache employing multiple rows or entries), generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)];

deducting the query size of each query removed from the cache size [since the size of the cache is the sum of the size of all queries in the tables, whenever a query is removed from the cache, the size of the cache decreases by the size of the query];

adding the new query to the inverse query engine cache [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache

(column 8, lines 26-29); First, Schneider teaches maintaining the inverse query engine cache at or below a maximum cache size as new items are added to the filter table [While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)] by trimming old, least used items from the table [the method continues to execute and maintain the cache at maximum size; generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)]. Thus maintaining the inverse query engine cache at or below a maximum cache size includes both adding new entries and removing old entries. Second, since Schneider's method continues to execute and maintain the cache at maximum size (column 8, lines 26-29), it must continuously monitor and keep track on the total size of cache, which requires the monitoring how many entries have been added and how many entries have been removed in order to update the total size]; **and**

adding a new query size to the cache size, the new query size identifying a size of the new query added to the inverse query engine cache [since the size of the cache is the sum of the size of all queries in the tables, whenever a query is added to the cache, the size of the cache increases by the size of the query];

wherein the inverse query engine cache comprises a control module, a cache [it is inherent that the query engine cache has a cache, otherwise it will not be a cache], **an add filter module** [Schneider: While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)], **a**

remove filter module [Schneider teaches any unique items (i.e., values) are added to the cache (column 8, lines 26-29)) by trimming old, least used items from the table (generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28); Chen: paragraph 0178: filter removal]**, a matcher** [Schneider: The principle advantage is if a cache hit (i.e., lookup with appropriate match) occurs, the system has saved time which would otherwise be required for executing the subquery to resolve a subquery result value (column 6, lines 49-53); Chen: All notifications matching the subscription can be delivered to the subscriber via a callback or other type of function that the subscriber provides at the time it registers its subscription or at other times. One subscription can be broken down into many filters (paragraph 0039)]**, a maintainer** [Schneider: While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)]**, an expire module** [Chen: paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired]**, a trim module** [Schneider teaches any unique items (i.e., values) are added to the cache (column 8, lines 26-29)) by trimming old, least used items from the table (generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28); Chen: paragraph 0178: filter removal]**, a cache weight module** [Schneider: In particular, a subquery cache is

provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29); Chen: paragraph 0156: 3) filters may involve fixed size or variable-size attributes], **a cache weight** [Schneider: In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29); Chen: paragraph 0156: 3) filters may involve fixed size or variable-size attributes], **an optimal weight** [Schneider: In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)], **a maximum weight** [Schneider: In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract); While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)], **a filter**

table [$\boxed{\text{Schneider}}$: the query engine (figure 2, 260) contains filter tables (Tables, figure 2, 250; Tables T1 and T2, figure 3A, 310 and 320); $\boxed{\text{Chen}}$: abstract: a filter receiving module, a filter reduction module, and a filter propagation module]**, a most recently used list** [$\boxed{\text{Chen}}$: Intelligent router 92 time marks the data (step 324) and locally caches it such as in memory 94 or secondary storage 97 (step 326). It indexes the cached data by, for example, channel ID, subjects, and time stamps (step 328) (paragraph 0100); It should be noted that the time stamps information provides directly the most recently used as well as the least recently used attributes; $\boxed{\text{Schneider}}$: generally cache rows are filled on a least-recently used_(LRU) basis (column 7, lines 26-28)]**, and an expiration list** [$\boxed{\text{Chen}}$: paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired];

wherein the expiration list comprises a filter identifier including an expiration value in an expiration field [$\boxed{\text{Chen}}$: paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired];

removing a filter based on an expiration time [$\boxed{\text{Chen}}$: paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the

filter to see if the filter has expired; thus a filter is removed (i.e. not propagated any further) if it has expired];

trimming the filter table [Schneider teaches maintaining the inverse query engine cache at or below a maximum cache size as new items are added to the filter table (While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)) by trimming old, least used items from the table (generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28); Chen: paragraph 0178: filter removal]; **and**

wherein the inverse query engine cache is used exclusively by an inverse query engine to store filters associated therewith [Schneider: Database system and methods are described for improving execution speed of database queries (e.g., for decision support) by optimizing execution of nested queries or "subqueries," such as are commonly used in client/server database environments. In particular, a subquery cache is provided having a size which can be dynamically adjusted by the system during execution of the query, for achieving an optimal cache size (abstract)].

Regarding claim 17, Schneider does not explicitly mention **an expire module**, **an expiration list**, and **removing a filter based on an expiration item.**

However, Chen teaches these limitations [paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter

time-out. In this case, the determining step 458 checks the filter to see if the filter has expired; paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired; thus a filter is removed (i.e. not propagated any further) if it has expired].

Therefore, it would have been obvious for one of ordinary skills in the art to recognize the need to remove the contents of a memory block when the size of the memory is limited, as demonstrated by both Schneider and Chen, and that removing the contents of a memory block based on an expiration time is a straightforward and efficient way of achieving this goal, as demonstrated by Chen, and to incorporate it into the existing system disclosed by Schneider, to further enhances the cache replacement strategy beyond the commonly adopted "Least Recently Used" scheme, and makes better utilization of the precious resource of cache capacity.

Regarding claim 17, Schneider does not explicitly mention the use of **a most recently used list**.

However, Schneider does teaches the use of Least Recent Used method for removing an item from the filter table [generally cache rows are filled on a <u>least-recently used (LRU) basis</u> (column 7, lines 26-28)].

Further, Chen teaches time marks the data to be cached [Intelligent router 92 <u>time marks the data</u> (step 324) and locally caches it such as in memory 94 or secondary storage 97 (step 326). <u>It indexes the cached data</u> by, for example, channel ID, subjects, and <u>time stamps</u> (step 328) (paragraph 0100)]. It should be noted that the

time stamps information provides directly the most recently used as well as the least recently used attributes.

Therefore, it would have been obvious for one of ordinary skills in the art to include the time stamps information, as demonstrated by Chen, and to incorporate it into the existing system disclosed by Schneider, so that the cache controller would have the most recently used information in addition to the least recently used information, which would further enhance the replacement/removing scheme of a filter entry.

As to claim 18, Schneider teaches that **the method as recited in claim 17, wherein the removing step further comprises removing a query from the inverse query engine that has been used less recently than other queries stored in the inverse query engine cache** [generally cache rows are filled on a least-recently used (LRU) basis (column 7, lines 26-28)].

As to claim 19, refer to "As to claim 17" presented earlier in this Office Action.

As to claim 20, Schneider teaches that **the one or more computer-readable media as recited in claim 17, wherein the step of adding the new query size to the cache size is performed before determining if the cache size is greater than or equal to the maximum cache size** [figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row; While the number of items is less than the cache size (step 401), the method continues to execute and maintain the cache at maximum size; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 21, refer to "A to claim 6" presented earlier in this Office Action.

As to claim 22, Schneider teaches that **the one or more computer-readable media as recited in claim 17, wherein the new query size is received with the new query** [figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row; While the number of items is less than the cache size (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 23, Schneider teaches that **the one or more computer-readable media as recited in claim 17, further comprising instructions to perform the additional step of determining the new query size** [figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row; While the number of items is less than the cache size (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 24, Schneider teaches that **the one or more computer-readable media as recited in claim 23, wherein the determining the new query size further comprises estimating the new query size** [figure 4A, step 401, "while items < cache rows" determines if the size of the items to be added would exceed the size of a cache row; While the number of items is less than the cache size (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 25, refer to "As to claim 13" presented earlier in this Office Action.

As to claim 26, Schneider teaches that **the one or more computer-readable media as recited in claim 17, wherein a query size is represented as a weight value that denotes the relative size of the query associated therewith with regard to other queries stored in the inverse query engine cache** [the corresponding cache weight is the sum of the size of all filter items currently stored in the cache; While the number of items is less than the cache size (step 401), the method continues to execute and maintain <u>the cache at maximum size</u>; any unique items (i.e., values) are added to the cache (column 8, lines 26-29)].

As to claim 27, refer to "A to claim 1" presented earlier in this Office Action.

As to claim 28, refer to "As to claim 6" presented earlier in this Office Action.

As to claim 29, Chen teaches that **the method as recited in claim 28, wherein an expired filter is a filter that has been stored in the inverse query engine cache for a predefined period of time** [paragraph 0156: 21) time-out parameters for filter; paragraph 0151: 2) filters should expire after the subscribers have moved; paragraph 0154: 16) expiration of filters; paragraph 0158: 3) filter persistency (e.g., expiration time-outs); paragraph 0176: filter sender may have included a filter time-out. In this case, the determining step 458 checks the filter to see if the filter has expired; thus a filter is removed (i.e. not propagated any further) if it has expired].

As to claim 30, refer to "As to claim 29" presented earlier in this Office Action.

As to claim 31, refer to "As to claim 7" presented earlier in this Office Action.

As to claim 34, refer to "As to claim 6" presented earlier in this Office Action.

As to claim 35, refer to "As to claim 29" presented earlier in this Office Action.

As to claim 36, refer to "As to claim 29" presented earlier in this Office Action.

As to claim 38, refer to "As to claim 13" presented earlier in this Office Action.

9. ***Related Prior Art***

The following list of prior art is considered to be pertinent to applicant's invention, but not relied upon for claim analysis conducted above.

- DeMarcken et al., (US Patent Application Publication 2004/0249682), "Filling a Query for Travel Planning."
- Haas et al., (US 6,934,699), "System and Method for Loading a Cache with Query Results."
- Gupta et al., (US 7,035,846), "Method, Computer Programs and Apparatus for caching Directory Queries."
- Jackson, (US Patent Application Publication 2003/0123387), "Device and Method for Filtering Network Traffic."
- Bennett, (US Patent Application Publication 2003/0204664), "Cache with Multiway Steering and Modified Cycle Reuse."
- Fu et al., (US Patent Application Publication 2004/0111519), "Access Network Dynamic Firewall."

Conclusion

10. Claims 1-38 are rejected as explained above.

11. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

12. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Sheng-Jen Tsai whose telephone number is 571-272-4244. The examiner can normally be reached on 8:30 - 5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Matthew Kim can be reached on 571-272-4182. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Sheng-Jen Tsai
Examiner

Art Unit 2186

November 19, 2007

MATTHEW KIM
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100